Knowledge Technologies Notes

Topic: 1 Introduction

Uses of Computation:

Concrete Tasks: (crack a code, compute object trajectory, operate camera view)

- Well-defined
- Can assess whether solution is correct
- Data is transformed in a mechanical way
- Only enhance human knowledge in a limited way.

Knowledge Tasks: (web query response, language translation, find optimal route)

- Data is irregular or unreliable.
- Outcome is not well-defined.
- Use, produce or enhance human knowledge.

E.g translation is not well defined because it leads to a loss of a meaning.

Classification of problems in to 5 Categories:

- Simple: Relationship between cause and effect is clear.
- Complicated: Cause and effect relationship requires analysis.
- Complex: Cause and effect relationship only realized after event.
- Chaotic: No obvious relationship between cause and effect.
- Disorder: Not knowing what type of causality exists.

Knowledge Technologies:

- Technologies tend to be either fairly general or fairly specific.
- General: Data must be transformed to suit the axioms or assumptions of the method, in a rigorous way. (find features of data and categorize)
- Specific: Detailed understanding of the task is used to drive development of the method, may use solutions to similar problems. (parse a language)

Data: Various sensor readings/variables/record values.

Knowledge: Interpretation of data, to help solve a problem.

Topic: 2 String Search

Lecture notes introduce regular expressions. This is a concept that is best learnt through practice. Just practice solving different regular expression questions.

Topic: 3 Approximate Matching

Two main applications for approximate string search:

- Spelling correction
- Computation genomics

Spell correction:

Dictionary: List of entries that are 'correct'.

Neighborhood Search:

- Generate all variants of w that utilize at most k changes (Insertions/Deletions/Replacements) – neighbors.
- Check whether generated variants exist in dictionary.
- All results found in dictionary are returned.

Consider: alphabet size is $|\sum|$, length of string is $|w|$:

For k edits, roughly $O(|\sum|^k \times |w|^k)$ neighbors

But $\sum$ is a small constant, string of interest is usually short, and k is usually small

For each neighbor, need a dictionary read (dict had D entries): Binary search yields $O(|w|^k \log D)$ string comparisons.

Global Edit Distance:

Scan through every dictionary entry looking for the 'best' match. Transform the string of interest into each dictionary entry, using insert, delete, match.

Lavenshtein distance: Match (0), Insert(+1), Delete(+1), Replace(+1)

Needleman-Wunsch Algorithm Pseudocode:

From string f to string t, given array A of $|f| + 1$ columns and $|t| + 1$ rows, we can solve using the Needleman–Wunsch algorithm: As the algorithm progresses, the Aij will be assigned to be the optimal score for the alignment of the first j characters in f and the first i characters in t.

```
lf = strlen(f); lt = strlen(t);
A[0][0]=0;
for (j=1; j<=lt; j++) A[j][0] = j * i;
for (k=1; k<=lf; k++) A[0][k] = k * d;
for (j=1; j<=lt; j++)
   for (k=1; k<=lf; k++)
      A[j][k] = max3(//Or min3 if m < i, d, r
         A[j][k-1] + d, //Deletion
         A[j-1][k] + i, //Insertion
         A[j-1][k-1] + equal(f[k-1],t[j-1]));//Replace or match
```

Note: m= cost of match; r = cost of replace; i = insertion cost; d = delete cost

Local Edit Distance:

If the m > i, d ,r then when you get any negative numbers, you convert to 0.

Smith-Waterman Algorithm Pseudocode:

From string f to string t, given array A of $|f| + 1$ columns and $|t| + 1$ rows, we can solve using the Smith–Waterman algorithm:

```
lf = strlen(f); lt = strlen(t);
A[0][0]=0;
for (j=1; j<=lt; j++) A[j][0] = 0;
for (k=1; k<=lf; k++) A[0][k] = 0;

for (j=1; j<=lt; j++)
   for (k=1; k<=lf; k++)
      A[j][k] = max4( //Or min4 if m<i,d,r
         0,
         A[j][k-1] + d, //Deletion
         A[j-1][k] + i, //Insertion
         A[j-1][k-1] + equal(f[k-1],t[j-1]));//Replace or match
```

`equal()` returns m if characters match, r otherwise

Final score is greatest value in the entire table (or least value, if $m < i, d, r$)

Edit Distance Complexity:

Complexity: O(|f||t|) in both space and time.

When approximate matching, we have a constant string f which we compare to each string t in the dictionary D:

$O(|f|\sum_{t \in D}|t|)$

N-Gram Distance:

$|Gn(s)| + |Gn(t)| - 2 * | Gn(s) \cap Gn(t)|$

- The greater the N value, the more constrained matches rare, so there has to be more consecutive matching -> higher accuracy but then you get less matches.
- On the other hand of N value is too, small you have a lot of nonsense n-grams matching with everything.
- More sensitive to long substring matches, less sensitive to relative ordering of strings (matches can be everywhere).
- Quite useless for very long strings and small alphabets.

Orthography (spelling) Vs Phonetics (sounds):

Soundex:

One mechanism: Soundex

Translation table:

aehiouwy	→	0 (vowels)
bpfv	→	1 (labials)
cgjkqsxz	→	2 (misc: fricatives, velars, etc.)
dt	→	3 (dentals)
l	→	4 (lateral)
mn	→	5 (nasals)
r	→	6 (rhotic)

Four step process:

1. Except for initial character, translate string characters according to table
2. Remove duplicates (e.g. 4444 → 4)
3. Remove 0s
4. Truncate to four symbols

Evaluation Metrics for spell correction:

Accuracy: Number of correct predictions / total number of words

Precision: Fraction of correct responses / total attempted responses

Recall: Number of words which have correct attempt / total number of words

Summary:

- What is approximate string search?
- What are some common applications of approximate string search; why are they hard?
- What are some methods for finding an approximate match to a string? What do we need to generate them?
- How can we evaluate a typical approximate matching system?

Topic: 3 Information Retrieval

Data Retrieval:

- Information is unambiguous.
- Queries are well defined: select * from Student where name = “bob”
- Data is stored in a logical form.
- **Used to retrieve items based on facts that describe them: e.g get articles from the Age dates 11/8/2017.**

Information Retrieval:

- Deals with the storage and retrieval of **relevant** documents.
- Emphasis on the **user** and the value of **content**.
- Real world objects, may not be consistent in format, language or length.
- A documents relevance is subjective to the users needs/wants.
- Information is ambiguous.
- Used to retrieve items based on meaning: e.g Is NZ a good holiday place.
- User has **information need**, user makes a **query**, IR engine **retrieves** documents.

Document: Text, Images, Music, Speech, Handwriting, Video, Genomes.

Document to be **returned** for a query: Document matches query in relevance.

Document to be **relevant** for a query: A document that matches a query well, may not actually be relevant to the end user. So, it doesn’t satisfy information need.

Information Needs:

- An IR system is used by someone who has an information need.
- Example: Do I want to move top Adelaide/ When is the next MC train.
- **Request for Information: “**global warming”
- **Factoid:** “What is the melting point of lead”
- **Topic Tracking:** “What is the history of this news story”
- **Navigational:** “University of Melbourne” (Navigating the web space)
- **Service or Transaction:** “Macbook Air”
- **Geospatial: “**Carlton restaurant” (Going from A to b)

What is Boolean Querying:

- For the query `diabetes` AND `risk`
- Take the bit representations `diabetes` = **110** `risk` = **011**
- Perform bitwise AND, $\wedge$, resulting in **010**
- Therefore document 2 is the only match

	doc1	doc2	doc3
juvenile	1	0	0
diabetes	1	1	0
risk	0	1	1
factor	0	1	1

To support:

- disjunction, simply use bitwise OR, $\vee$
- negation, use bitwise complement, $\hat{}$

`diabetes` AND ((NOT `risk`) OR `juvenile`)
110 AND ((NOT **011**) OR **100**) = **100**

Repeatable: The same query will always output the same result, if db unchanged.

Auditable: Ability to check exactly what action was performed.

Controllable: Can control on what answers come out based on conditionals.

Boolean querying however has no ranking system, there is also no control over result set size and it is difficult to make good heuristics/do well.

What is Ranked Querying:

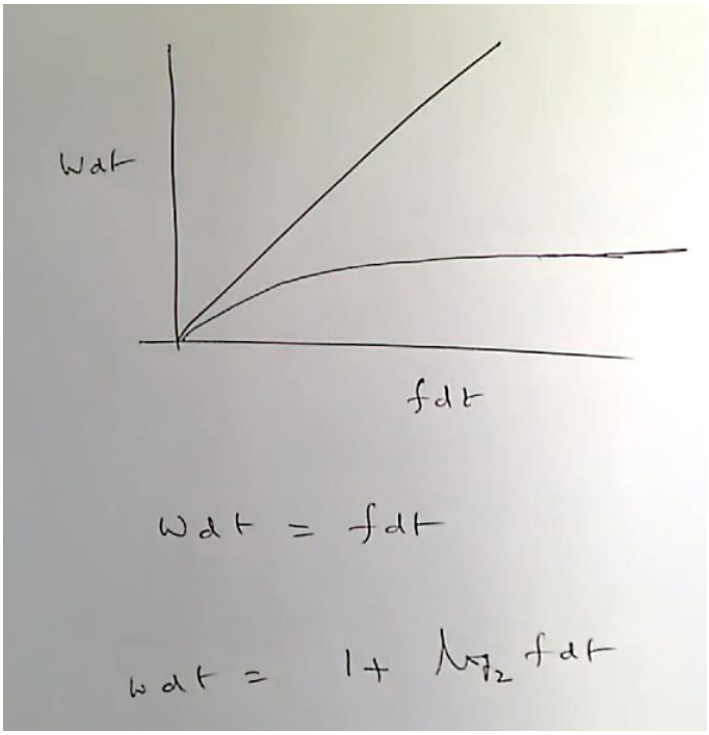
Documents that are returned for a given query are **ranked** based on their **similarity** and **relevance** to the query. More related to information retrieval.

What are the components of the TF-IDF Model:

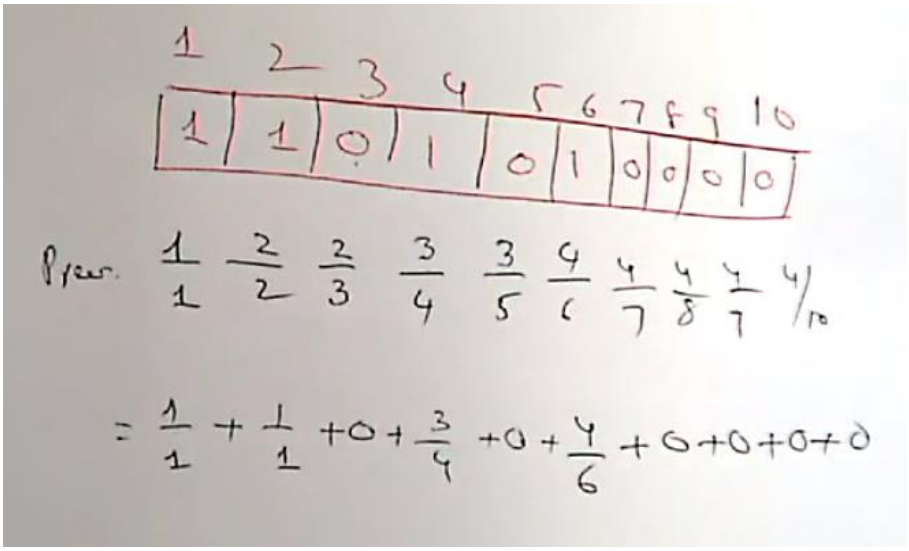
- **Inverse document frequency or IDF**: Less weight is given to a term that appears in many documents.
- **Term frequency or TF**: More weight is given to a document where a query term appears many times.
- Less weight is given to a document that has many terms. (long document)

The Vector-Space Model:

Can create a weighted vector for every document. Documents that are similar would share a similar spot in the vector space.



Reducing dominance of very frequent term.



How many relevant documents do I have in my returned values?

That sum / 4 is the average?

Topic: 4 Information Retrieval

Elements of a Web Search Engine:

Crawling: Search data is gathered from web

Parsing: Data translated into canonical form

Indexing: Data structures built to allow search to take place efficiently

Querying: Data structure processed in response to queries

1.Crawling Fundamentals:

- For document to be queried search engine must discover it by crawling.
- Attempt to visit every page of interest and retrieve for processing.
- **Challenges:** Some websites are short lived, other non-crawlable.
- **Assumption:** If a web page is of interest there will be a link to it from another page.

Crawling algorithm:

1. Create a prioritised list L of URLs to visit, and a list V of URLs that have been visited and when.
2. Repeat forever:
    1. Choose a URL u from L and fetch the page $p(u)$ at location u.
    2. Parse and index $p(u)$, and extract URLs $\{u'\}$ from $p(u)$.
    3. Add u to V and remove it from L. Add $\{u'\} - V$ to L.
    4. Process V to move expired or 'old' URLs to L.

Note: L should be prioritized to ensure: every page is visited eventually, significant or dynamic pages visited frequently, don't fall in crawler trap or infinite loop.

Crawler trap: E.g next month link on website can go on forever.

2.Parsing / Page Recognition / Page Analysis:

- Words of page extracted and added to data structure. Languages identified. Format of page is determined. Capture basic data such as metadata.
- **Concerns:** Avoid indexing invisible content, websites try to spoof data for SEO ranking boost.

Tokenization:

- Aim of parsing is to reduce a web page or query to a sequence of tokens.
- Good tokenization → query matches web pages without approx. matching.
- **Issues:**
    - **Hyphenation:** under-coating 1 word or two words?
    - **Compounding:** football 1 word or two words?
    - **Possessives:** Zadek's or Zadek what do we match with?

Canonicalization:

Indexing process that relies on fact extraction may need info in canonical form:

- Variant spelling color/colour
- Numbers: 1,800 vs 1800 vs one thousand eight hundred
- Dates: 5/4/2011, 4/5/2011, 5th of April 2011

Basically, we want a standard form that expresses all the variants.

Historically discarded: stop words such as 'the' 'or' 'and'

Stemming (Canonicalization):

- The most significant form of canonicalization.
- Different suffixes and prefixes for same word. Index the stem of words rather than full words. Stemming involves removing affixes.
- **Inflectional morphology:** how word derived from stem. E.g in+expense+ive leads to inexpensive.
- Sess -> ss, ies -> I, ational -> ate, tional ->tion

Zoning (**Canonicalization**)

Segment a document into zones such as title, anchor text, headings etc. Find weights for each zone. Also can make zones based on font size -> significance of text. Then e.g web engine can favor pages that have query terms in titles.

3. Indexing

Index: A data structure that maps terms to documents that contain them.

Inverted Index: A collection of lists, one per term, recording the identifiers of the documents containing that term.

Search structure

For each distinct word t, the search structure contains:

- A pointer to the start of the corresponding inverted list.
- A count f_t of the documents containing t.

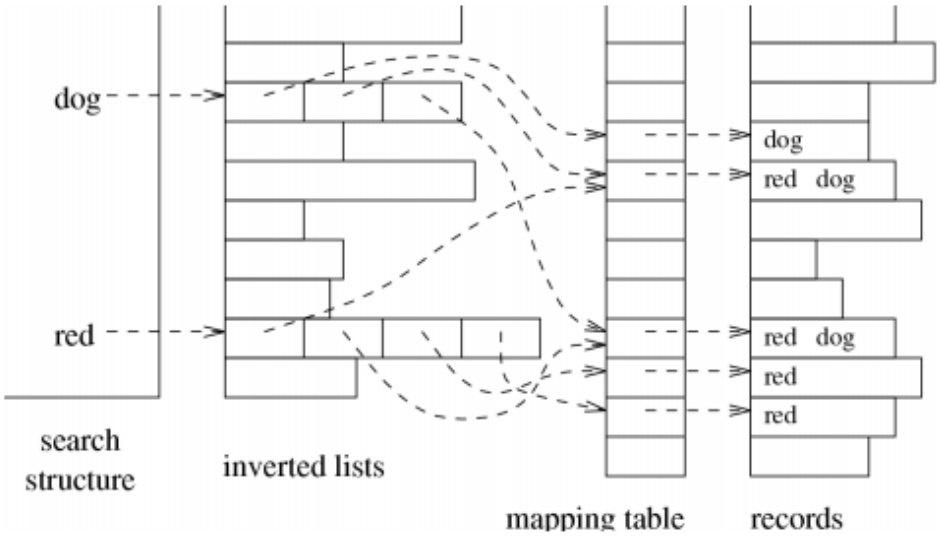
That is, the search structure contains the *vocabulary*.

Inverted lists

For each distinct word t, the inverted list contains:

- The identifiers d of documents containing t, as ordinal numbers.
- The associated frequency $f_{d,t}$ of t in d. (We could instead store $w_{d,t}$ or $w_{d,t}/W_d$.)

Search struct – hash map inverted list – linked list



Faster querying: 1. Terms in query correspond to search struct, 2. Index only indicates documents where the term is present. Index structs -> storage x 1.4

3. Querying

Boolean Querying:

- **Using Term document matrix (TDM):** Compact to store, bitwise comparisons are fast.
- **Using Inverted Index:** Fetch inverted list for each query term, use intersection of lists to resolve AND. Use union of lists to resolve OR. Compliment of list to resolve NOT. Ignore within document frequencies.

	Docs							
Terms	1	2	3	4	5	6	7	8
data	1	1	0	0	2	0	0	0
examples	0	0	0	0	0	0	0	0
introduction	0	0	0	0	0	0	0	0
mining	0	0	0	0	0	0	0	0
network	0	0	0	0	0	0	0	0
package	0	0	0	1	1	0	0	0

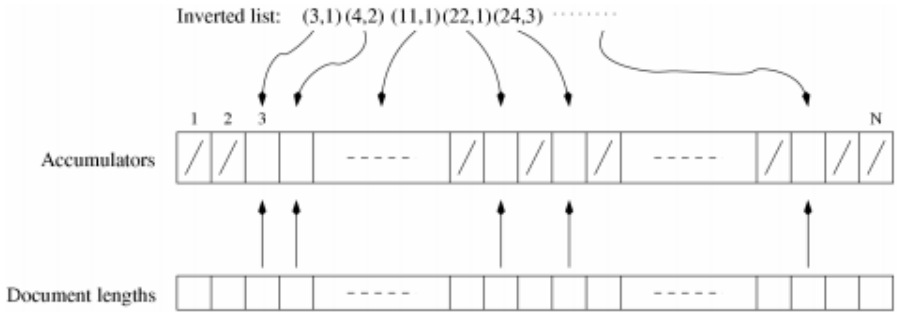
Ranked Querying:

To produce ranking for a typical TF-IDF model, with cosine similarity, we need:

- The frequency of each term in each document (TF)
- Number of documents where query term occurs (DF)
- Length of each document

Ranked Querying using Inverted Index:

1. Allocate an accumulator A_d for each document d, and set $A_d \leftarrow 0$.
2. For each query term t,
    1. Calculate $w_{q,t}$, and fetch the inverted list for t.
    2. For each pair $\langle d_t, f_{d,t} \rangle$ in the inverted list
        Calculate $w_{d,t}$, and
        Set $A_d \leftarrow A_d + w_{q,t} \times w_{d,t}$.
3. Read the array of W_d values and, for each $A_d > 0$,
    Set $A_d \leftarrow A_d / W_d$.
4. Identify the r greatest A_d values and return the corresponding documents.



That is, starting with a set of *N* zero'ed accumulators, use the lists to update the accumulators term by term.

Then use the document lengths to normalize each non-zero accumulator.

Accumulator Cost:

- Most accumulators are non-zero and an array is the most space and time efficient structure.
- But the majority of those accumulator values are trivially small.
- Can impose a limit on the number on accumulators, but not accumulating low ft terms.

Note: Check slides if you are interested in alterations to the algorithm.

Querying Costs:

Disk space: 40% of the size of the data

Memory space: Accumulators for the vocabulary and for caching previous results

CPU time: For processing inverted lists and updating accumulators

Disk traffic: Fetch inverted lists.

Phrase Queries:

Strategies for phrase query evaluation:

- Process queries as bags of words so terms occur anywhere in matching documents.
- Add word positions to the index entries so the location of each word in each document can be used during query evaluation. (Ignore phrases then retrieve potential matches and scan for phrases in documents)
- Use some form of phrase index or word pair index so they can be directly identified without inverted index.

Evaluation of query: the lord mayor of Melbourne

Could involve intersecting the lists for lord, mayor and Melbourne looking for positions p of lord such that mayor is at p+1 and Melbourne at p+3

Favor documents where terms near to each other. Search for phrases where terms are within a specified distance.

Link analysis:

In web search a strong piece of evidence for a page's importance is given by links. In particular how many other pages have links to this page.

Page Rank (link analysis)

- Each document has a fixed number of credits associated with it, a proportion which it redistributes to documents it links to; in turn, it receives credits from pages that point to it.
- The final n umber of credits the page is left with determines its pagerank.
- The process used to find the pagerank is determined by:
    - Random walk: Each page has same probability of being starting point for random walk
    - Teleport: Teleport to random page. For teleports and traversals of outgoing links, all pages have equal probability of being visited.

A High-Performance Web Search Engine:

- Cache answers to common queries.
- Index selected phrases.
- Use multiple server with caching
- Have separate servers for crawling and index construction.
- Integrate resources such as maps and directories.

Summary:

- Search involves crawling, parsing, indexing, and querying; practical search also involves a range of other technologies.
- Crawling is in principle a straightforward application of queuing, but practical issues mean that implementation is complex.
- Parsing involves discarding metadata and hidden information; tokenization; canonicalisation; zoning; and stemming.
- Inverted indices describe text collections as lists of the pages with each word, rather than the list of words on each page.
- The same structure is used for Boolean and ranked querying.
- Approximations can be used to reduce querying costs, which can affect the answer set in unpredictable ways.
- On the web, link and anchor information can be the dominant evidence of relevance.

Lecture Set 1: Machine Learning and Data Mining

The course content will be as follows:

- ❑ Introduction to Machine Learning/ Data Mining motivating when you have to use such techniques
- ❑ Basic introduction to probability theory
- ❑ Basic machine leaning techniques
  - NB
  - Decision Trees
  - SVMs
- ❑ Pattern mining and how they can be used
  - Frequent pattern mining
  - Association rule
  - Feature selection
- ❑ Clustering techniques
- ❑ Evaluation measures of various machine learning techniques: E.g. Accuracy, F-measure, AUC and Cross validation methods.

Introduction to Machine Learning/ Data Mining:

- Exponential Growth of data with the internet and data collection/generation.
- Extremely rich and complex datasets in life sciences: Genomics, gene sequencing.
- Need to somehow draw meaning and form meaning from the data.
- Need to process complex data.
- Applies to many fields: Bioinformatics, commerce, engineering, info retrieval

Research Issues:

- Data management/ Databases/ info retrieval

  Issues: Scale, data quality, incomplete, inconsistent. Need to answer use queries.

- Machine Learning and Data Mining

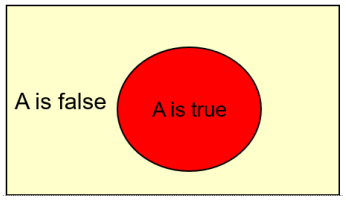
  Issues: Scale, dimensionality, data modalities. Need to discover models that fit data.

Lecture Set 2: Introduction to Basic Probability

Discrete Random Variables:

Takes on a countable number of values.

A is a Boolean-valued random variable if A denotes an event, and there is some degree of uncertainty as to whether A occurs. Uncertainty measured by probability.

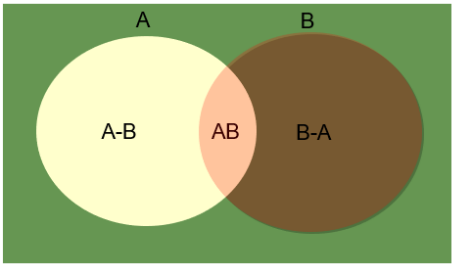


P(A) = probability that A is true

P(True) = 1 e.g P(A=head or A =Tail)

P(False) = 0 e.g (PA = head and A =Tail) or P(Head and Tail)

P(A or B) = P(A) + P(B) – P(A and B)



P(A) = P(A and B) + P(A and Not B)

P(A|B) = P(A and B)/P(B)

P(A and B) = P(A|B)*P(B) **(Product rule)**

P(A) = $\sum_B$ P(A|B)*P(B) **(Sum rule)**

Bayes Theorem:

P(B | A) = P(B,A) / P(A) → P(B|A) = P(A|B)*P(B) / P(A)

$$\mathbf{P(B|A) = P(A|B)*P(B) / \sum_B P(A|B)*P(B)}$$

Negation:

P(A) + P(~A) = 1

P(A) = P(A,B) + P(A, ~B)

P(A|B) + P(~A|B) = 1 But: P(A|B) + P(A|~ B) ~= 1

Example:

B = restaurant is bad ~ B = restaurant is good

S = menu is smudged

P(B) = 0.5 (**prior probability:** probability without any evidence/influence)

P(S|B) = 0.75 (likelihood)

P(B|S) is the **posterior probability:** probability based on evidence

P (B|S) = P (B,S)/p(S) = P (B, S)/[(P (S, B) + P (S,~B)]

= P (S|B)P(B)/[P (S|B)P(B) + P (S|~B)P(~B)] (**Bayes theorem)**

= (3/4)*(1/2)/[(3/4)*(1/2) + (1/3)*(1/2)]

= 9/13 =0.69

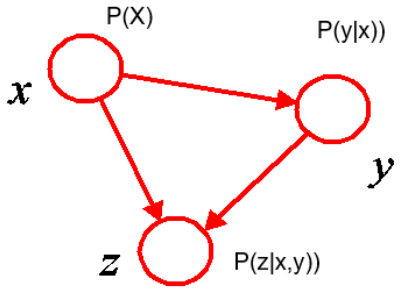
Probabilistic Graphical Models (PGM)

PGM provides new insights into existing models

Consider an arbitrary joint distribution

By successive application of the product rule $p(x, y, z)$

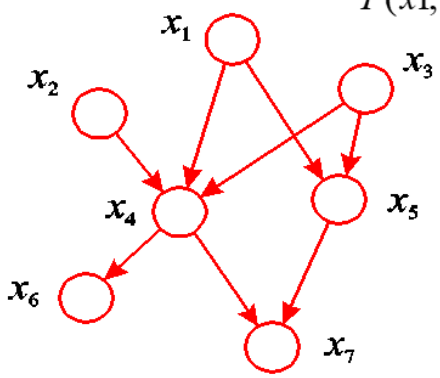
$$\begin{aligned} p(x, y, z) &= p(x)p(y, z|x) \\ &= p(x)p(y|x)p(z|x, y) \end{aligned}$$



Directed Acyclic Graphs

Joint distribution
where pa_i **denotes the parents of i**

$$p(x_1, \ldots, x_D) = \prod_{i=1}^{D} p(x_i|\text{pa}_i)$$



$$P(x1, x2, x3, x4, x5, x6, x7) = \begin{cases} P(x1) * P(x2) * p(x3) * \\ P(x4 \mid x1, x2, x3) * \\ P(x5 \mid x1, x3) * \\ P(x6 / x4) * \\ p(x7 \mid x4, x5) \end{cases}$$

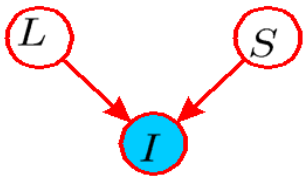
Conditional independence for directed graphs is similar, but with one subtlety

Illustration: pixel colour in an image

$$p(I,L,S) = p(L,S)p(I|L,S)$$

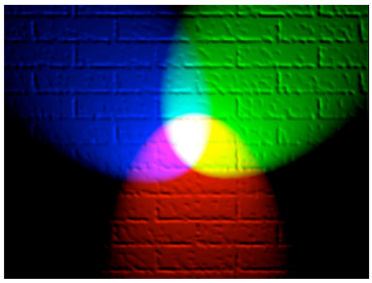
$$p(I,L,S) = p(L)p(S)p(I|L,S)$$

lighting colour of the room



surface colour of the painting

image colour



$$p(L, S) = p(L)p(S)$$

$$p(L, S|I) \neq p(L|I)p(S|I)$$

$p(I,L,S) \neq p(I)*p(L|I)*p(S|I)$

Lecture Set 3: Data Mining

Data mining Goals:

- Handle large, high dimensional and complex data
- Discover interesting patterns in processes
- Discover knowledge that is useful/actionable

Datamining Tasks Involve:

Prediction Methods (supervised learning methods)

Use some known variables to predict unknown or future values of other variables. (classification, regression, deviation detection)

Description Methods (unsupervised learning)

Find human-interpretable patterns that describe the data. (Clustering, Association rule discovery, sequential pattern discovery)

Classification (supervised learning)

Given a collection of training data. Each item contains values/features that is mapped to a class. This data is used to train the prediction model. Then whenever the model is presented values/features of an object, it should aim to determine the class label for this object.

Usually training data is partitioned into training and test sets. The test set is used to see how good the model is. Precision and accuracy can then be benchmarked with the test outcomes.

Example: classifying store customers as buyers/non-buyers based on features.

Clustering (unsupervised learning)

Given data points, each having a set of attributes and similarity measure among them find clusters such that data points in one cluster are similar. Data points in separate clusters are less similar to one another. Similarity measures: Euclidian distance for continuous attributes.

Example: Clustering documents with relevant terms together. Info retrieval can use the clusters to return documents in similar cluster.

Association Rule Discovery:

Given a set of records each of which contain some number of items from a given collection, produce dependency rules which predict occurrences of an item based on occurrences of other items.

Confidence = #transaction containing milk and coke/#transactions containing milk

Support for rule = #transactions containing Milk and Coke/#transactions

Example: What items are bought together in store transactions. Can then put both items together in store.

Sequential Pattern Discovery

Given a set of objects, with each object associated with its own timeline of events, find rules that predict strong sequential dependencies among different events. Pattern discovery.

Example: Athletic apparel store: Purchase shoes, racket and racket ball -> purchase sports jacket. If these things happen in sequence, then customer will likely buy jacket.

Regression:

Predict a value of a given continuous valued variable based on the values of other variables, assuming a linear non-linear model of dependency.

Example: Predicting sales amounts of new products based on advertising expenditure.

Deviation/Anomaly Detection:

Detects signification deviations from normal behavior.

Example: credit card fraud detection.

Challenges of Data Mining: Scalability, data quality, protect privacy, Dimensionality, understandability of model, incremental learning.

Lecture Set 4: Introduction Classification

Given:

- A set of training tuples and their associated class labels.
- Each tuple X represents n dimensional attribute vector (X=x1,..xn)
- There are K classes C1, C2,..,Ck

Naïve Bayesian classifier:

Naïve assumption: Attributes in X are conditionally independent

For each class Ci, estimate probability p(Ci|X)

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})} \quad \left(posteriori = \frac{likelihood \times prior}{evidence} \right)$$

Predicts X belongs to Ci iff the probability P(Ci|X) is the highest among all the P(Ck|X) for all the K classes.

Since P(X) is constant for all classes, only P(Ci|X) = P(X|Ci)P(Ci) needs to be maximized.

Naïve Bayesian Classifier

$$\text{Storage required is } O(C-1+C\sum_{i=1}^{D}(Di-1))$$

$$= O(C-1-CD+C\sum_{i=1}^{D}Di)$$

$$= O(C(\sum_{i=1}^{D}Di-D))$$

$where$

$D = number \quad of \quad \dim ensions$

$Di = ith \quad domain \quad size$

$C = number \quad of \quad classes$

$$p(c_{i_|}|x) = p(ci) * \prod_{k=1}^{n} p(x_k|c_i)$$

- Good to apply before more sophisticated classification.
- Scales easily for large dimensions
- Easy to update decisions based on additional information.

Example:

Headache	Cough	Temperature	Sore	Diagnosis
severe	mild	high	yes	Flu
no	severe	normal	yes	Cold
mild	mild	normal	yes	Flu
mild	no	normal	no	Cold
severe	severe	normal	yes	Flu

P(FLU) = 3/5	P(Cold) = 2/5
P(Headache = severe | Flu) = 2/3	P(Headache = severe | Cold) = 0/2 ~= e
P(Headache = mild | Flu) = 1/3	P(Headache = mild | Cold) = 1/2
P(Headache = no | Flu) = 0/3 ~= e	P(Headache = no | Cold) =1/2
P(Sore = severe | Flu) = 1/3	P(Sore = severe | Cold) = 1/2
P(Sore = mild | Flu) = 2/3	P(Sore = mild | Cold) = 0/2 ~ e
P(Sore = no | Flu) = 0/3 ~ e	P(Sore = no | Cold) = 1/2
P(FLU) = 3/5	P(Cold) = 2/5
P(Temperature = High | Flu) = 1/3	P(Temperature = High | Cold) = 0/2 ~e
P(Temperature = Normal | Flu) = 2/3	P(Temperature = Normal | Cold) = 2/2
P(Cough = yes | Flu) = 3/3	P(Cough = yes | Cold) = 1/2
P(Cough = no | Flu) = 0/3 ~=e	P(Cough = no | Cold) = 1/2

e= small value = 10^{-7} (one can use e to be less than 1/n where n is the number of training instances)

P(Flu| Headache = severe, Sore = no, Temperature = normal, Cough = yes)

= P(Flu)*P(Headache = severe|Flu)*P(Sore = no|Flu)*P(Temperature = normal |Flu)*P(Cough = yes|Flu)

= 3/5 x 2/3 x e x 2/3 x 3/3 = 0.26e

P(Cold| Headache = severe, Sore = no, Temperature = normal, Cough = yes)

~ P(Cold)*P(Headache = severe|Cold)*P(Sore = no|Cold)*P(Temperature = normal |Cold)*P(Cough = yes |Cold)

= 2/5 x e x ½ x 1 x ½ = 0.1e

=> Diagnosis is Flu

Evaluating classifiers:

How to evaluate Classifiers?

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

Accuracy with respect to positive cases also called true positive rate

$$Specificity = \frac{TN}{TN + FP}$$

Accuracy with respect to negative cases

$$\text{Re}call = \frac{TP}{TP + FN}$$

$$\text{Pr}ecision = \frac{TP}{TP + FP}$$

$$F1\_Score = \frac{2\,\text{Re}call * \text{Pr}escision}{\text{Re}call + precision}$$

		Actual	
		P	N
Predicted	P	TP	FP
	N	FN	TN

False negative rate = FN/(TP+FN)

TP – True Positive FP – False Positive FN – False Negative TN – True Negative

Evaluation Schemes:

Leave-One-Out:

- If we have N data points, which we know the labels
- We choose each data point as a test case and the rest as training data.
- We do this N times, thus train the system N times. Find average.

Good:

- No sampling biases.
- Unique results and repeatable for different methods.
- Generally higher accuracy. Since N-1 points used in training.

Bad:

- Infeasible if we have large data set and training is expensive.

10-Fold Cross Validation:

- Assume we have N data points, we know the labels.
- Partition data into 10 equal size partitions.
- Choose each partition for testing and remaining 9 for training.
- Thus, we will train the system 10 times. Find average.

Good:

- Need to train the system only 10 times, unlike Leave-One-Out which requires N times.

Bad:

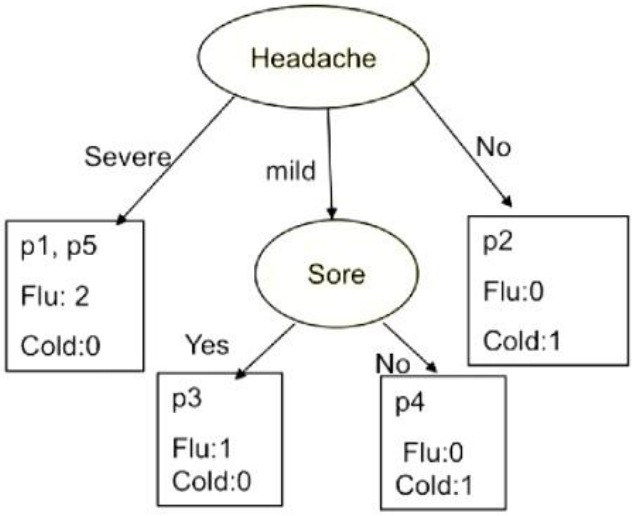
- There can be a bias in evaluating due to the way we partition.
- Results will not be unique unless we always partition the data identically. One solution is to repeat 10-fold cross validation by randomly shuffling the data by 5 or more times.
- Slightly lower accuracy since 90% of data is used for training.
- Smaller data sets are not always possible to partition. (so that each partition represents identically independently distributed)

Lecture Set 5: Decision Trees

Rule Based Classification:

Patient#	Headache	Cough	Temperature	Sore	Diagnosis
p1	severe	mild	high	yes	Flu
P2	no	severe	normal	yes	Cold
P3	mild	mild	normal	yes	Flu
P4	mild	no	normal	no	cold
p5	severe	severe	normal	yes	Flu

- Construct a decision tree.
- Extract one rule for each leaf node.
- Example:



Rule 1: (headache = severe) the it is flu

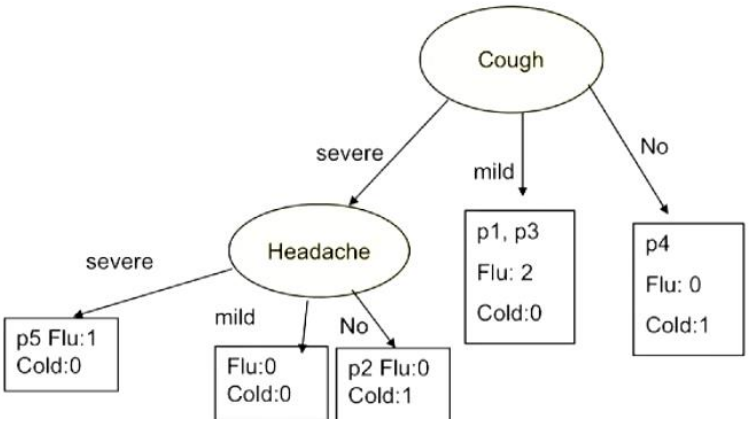
Rule2: (headache = mild) and (sore =yes) then it is flu

Rule 3: (headache = mild) and (sore = no) then it is cold

Rule 4: (headache = no) then it is cold

Note:

- You need discrete values to be able to make these trees. E.g severe, mild, no
- You can build different shapes of decision tress e.g:
- How to build the optimal decision tree? Maybe have purity in leaf nodes? Or maybe number of nodes in minimized.



- How many branches do we take, do we wait until pure or stop?

Complexity for finding optimal tree: exponential time

It is NP (non-deterministic polynomial) hard to make an optimal tree.

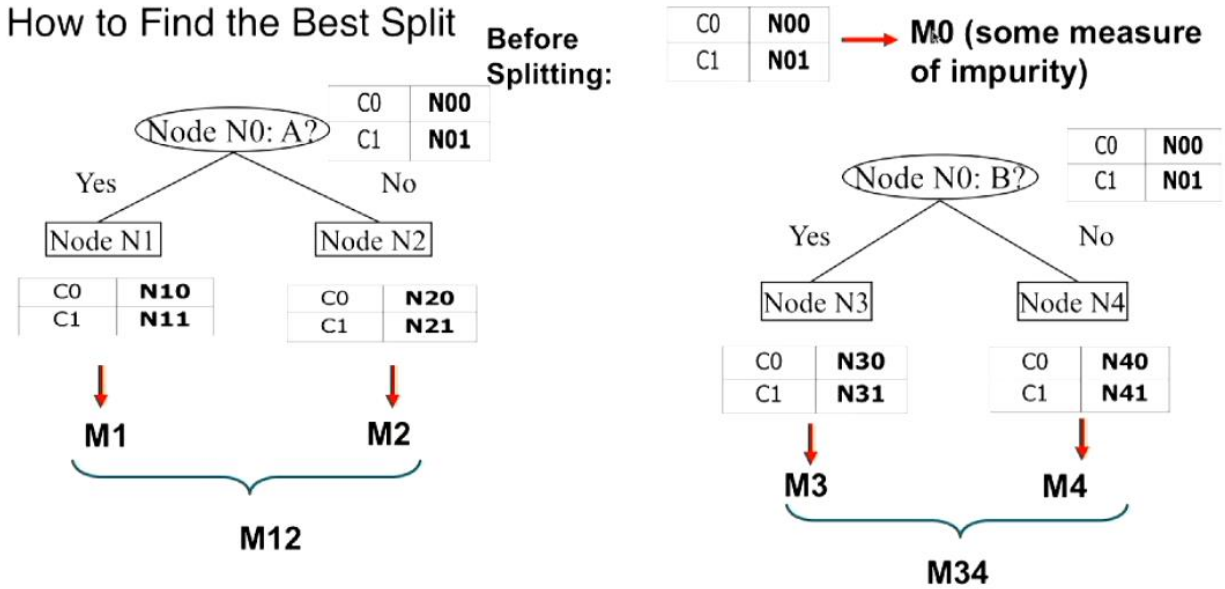
2^n possible trees, where n is the number of attributes. All possible set partitions.

Use Heuristics instead:

- Choose an attribute to partition the data at the node such that each partition is homogenous (least impure).
- Stop the growth of the tree if all the leaf nodes are largely dominated by a single class.
- Basically, measure the purity of every attribute used as a branch and pick the attribute with the best purity as the attribute to use next in the tree graph when building it.

Measures of Node Impurity:

- Gini Index
- Entropy
- Misclassification error



Gain = (M0 – M12) vs (M0 – M34)

Where M is some measure of impurity (discussed later).

-M1 is the measure of impurity of Node1

-M2 is the measure of impurity of Node2

-M12 is the average impurity of the M1 and M2, and is the impurity of N00 and N01

-Node N0 is for attribute A. So, if N0 is better than other nodes, then Attribute A is the better choice.

- So the gain basically compares the impurity of that results from picking a certain attribute.

GINI Index:

Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(Where *p(j | t)* is the *relative frequency* of class j at node t).

- Maximum value of Gini index = $(1 - 1/n_c)$ when records are equally distributed among all classes, implying least interesting information or most impure.
- Minimum is (0.0) when all records belong to one class, implying most interesting information or most pure or most homogeneous
- Examples:

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	2
C2	4
Gini=0.444	

C1	3
C2	3
Gini=0.500	

$1 - (0/6)^2-(6/6)^2 = 0$ $1- (1/6)^2-(5/6)^2 = 0.278$ $1- (2/6)^2-(4/6)^2= 0.444$ $1 - (3/6)^2 - (3/6)^2 = 0.5$

- Gini index is 0, that means its very pure. Perhaps c2 is flu and every patient belong to flu here. 0.5 is the largest gini index for binary.

Splitting Based on GINI

Used in CART, SLIQ, SPRINT.
When a node p is split into k partitions (children), the quality of split is computed as,

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,
n = number of records at parent node p.

If GINI(j) – GINI split (j) > delta then split the node j.

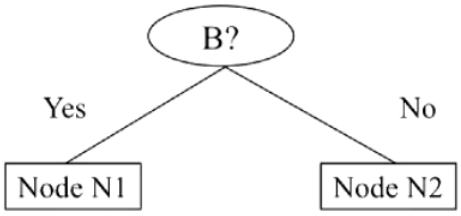
Using gini index to generate partition tree. Computer gini before and after splitting. If splitting makes gini index smaller, than that means that node is more pure.

Binary Attributes: Computing GINI Index

Splits into two partitions

Effect of Weighing partitions:

- Larger and Purer Partitions are sought for.



	Parent
C1	6
C2	6
Gini = 0.500	

Gini(N1)
= 1 – (5/7)² – (2/7)²
= 0.408

Gini(N2)
= 1 – (1/5)² – (4/5)²
= 0.32

	N1	N2
C1	5	1
C2	2	4
Gini=0.371		

Average Gini(Children)
= 7/12 * 0.408 +
5/12 * 0.32
= 0.371

The more branches you make, the higher the chance of a pure node.

Categorical Attributes: Computing Gini Index

For each distinct value, gather counts for each class in the dataset

Use the count matrix to make decisions if the parent node has instances: 5 Family; 3 Sports and 2 Luxury its Gini Index is 0.62.

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

Continuous Attributes: Computing Gini Index...

For efficient computation: for each attribute,

- Sort the attribute on values
- Linearly scan these values, each time updating the count matrix and computing Gini index at points where class label changes (at points A and B)
- Choose the split position that has the least Gini index

	Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Taxable Income										
Sorted Values →		60	70	75	85	90	95	100	120	125	220
Split Positions →		55	65	72	80	87	92	97	110	122	172	230
		<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >
	Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0
	No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0
	Gini	0.420	0.400	0.375	0.343	0.417	0.400	*0.300*	0.343	0.375	0.400	0.420

A

B

Can measure purity with entropy:

Less entropy means homogenous -> pure. So if everything belongs to one class then entropy is 0.

Alternative Splitting Criteria based on INFORMATION gain

Entropy at a given node t:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

(Where *p(j / t)* is the relative frequency of class j at node t).

- Measures homogeneity of a node.
  - Maximum (log n_c) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
- Entropy based computations are similar to the GINI index computations

Max: -Sum 1/n log(1/n) -> 1/n + 1/n.. n time * log((1/n)^-1) -> log n

Examples for computing Entropy

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Information Gain: $$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node, p is split into k partitions;

n_i is number of records in partition i

- Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
- Used in ID3 and C4.5 (Decision Tree software)
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

Splitting Based on INFO...

$$SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO}$$

Parent Node, p is split into k partitions

n_i is the number of records in partition i

- Adjusts Information Gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5
- Designed to overcome the disadvantage of Information Gain

- If each leaf node has only 1 item then entropy will be 0. Only 1 class. More splits you make, you divide by larger number to avoid doing too many splits.

Splitting Criteria based on Classification Error

Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

Measures misclassification error made by a node.

- Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

- P(i|t) should be maximized. I t is the class with the highest probability of occurring at a leaf node. 1 – maximum is basically the chance we make a mistake or error. This is why we want pure nodes so we have a lower chance of making errors

Examples for Computing Error

$$Error\ (t) = 1 - \max_i\ P(i \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

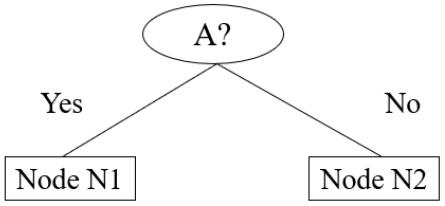
Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Misclassification Error vs Gini



	Parent
C1	7
C2	3
Gini = 0.42	
MissClass = 0.3	

Gini(N1)
= 1 – $(3/3)^2$ – $(0/3)^2$
= 0

Gini(N2)
= 1 – $(4/7)^2$ – $(3/7)^2$
= 0.489

MissClass(N1) = 1 –(3/3) = 0

MissClass(N2) = 1 – 4/7 = 3/7

	N1	N2
C1	3	4
C2	0	3
Gini = 0.342		
MissClass =0.3		

Gini(Children)
= 3/10 * 0
+ 7/10 * 0.489
= 0.342

Gini improves !!

MissClass =3/10 *0+(7/10)*(3/7)=0.3

Missclassification unchanged!

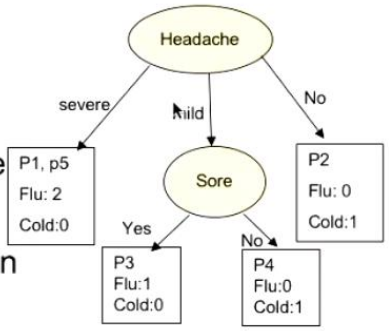
Misclassification says there is no reason to split this node. It won't improve error. But gini index says you should split. So different decision trees work differently on different data sets and so you have to use different decision trees and work out which one is the best for your data.

Lecture Set 6: Rule Based Classification and Random Forests

Rule based classification

Advantages of Rule synthesis based on Decision Tress are

- All rules are mutually exclusive this implies only one rule will be applicable at the time of decision making
- Easy to extract the rules from traversing the decision tree



- Disadvantages
  - Too restrictive in terms of number rules learned
  - The rules may be too specific and can be very complex
- There are many other schemes proposed in the literature and we cover them in the topic of frequent pattern mining.

e.g disadv1: more than 1 reason a person is suffering from flu.

e.g disadv2: you may have a very long branch

Variants of Decision Trees:

Decision Trees cannot capture full information available in the data. This is especially true for data sets with very large number of dimensions.

Solution 1: Bagging

Build several decision trees by sampling the data. Randomly select N records. We sample N records randomly with replacement. Continually pick items from the bag (adding item back to bag) until you made n picks then use these n to make a tree. On average each sample will have 2/3 of the N original records. We build a decision tree for each sample. Maybe 20-50 such trees?

Decision: At the time of classification we collect all the decision trees and apply majority rule (pick decision that most trees point to) to make the final decision. Generally, works well if trees independent.

Solution 2: Random Forests

Similar to bagging, but construction of each tree is different to the standard decision tree method:

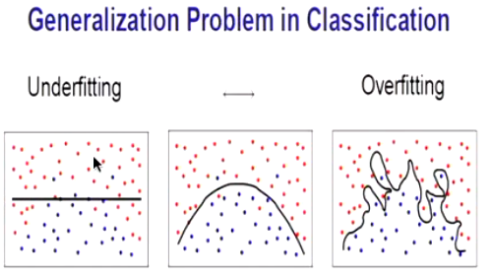
- At each node we first randomly select m features out of all the feature, unlike the standard method which considers all the non-chosen features so far and chooses the best feature to split the node among these attributes. **This makes the trees quite different from each other. Since randomly choosing features so errors independent from one another.**
- Unlike standard decision tree method, there is no need to prune (reduce size of tree) the constructed trees due to the randomness imposed both on the selection of data the tree constructions. Even better classification.
- **Best performance:** When the trees are no correlated to each other (independent). Overlap of splitting attributes between any two trees is small. This improves accuracy.
- **How to make trees random:**
    - Choosing randomly N record with replacement.
    - Selecting the best attribute for splitting from randomly chosen 'm' attributes for each splitting node.
- The quality of the decision tree depends on its accuracy which means large m value.
- The independence of each tree depends on the small values of m (chosen splitting variable at each node is likely different to other trees).
- Overall quality of the random forest depends on both accuracy of the trees and their independence. Thus, there needs to be an optimal m value which can be determined experimentally.

Practical Issues of Classification

Underfitting and Overfitting:

• For decision trees, we need to make sure it does not overfit the data.

• One way to do this is to have a stopping criterion that makes sure that if the GINI index is smaller than some threshold value we stop splitting the node.



• Build the tree completely such that no further splitting is possible. That is either the node is pure (contains one class) or all the attribute values identical (data is inconsistent – records having identical feature values are labeled differently). Once the tree is built we start pruning the tree using some statistical significance test – that is improvement in accuracy is significant.

Practical Issues of Classification

Missing Values:

Patient#	A1	A2	A3	A4	Diagnosis
p7	a11	?	a31	?	c1

- Missing values can be predicated by using the distribution of the attribute values. Especially if we find two highly correlated attributes we can use this correlation to predict the missing value.

• Costs of Classification:

- We need to optimize the decision based on cost involved using confusion matrix

Practical Issues of Classification

Costs of Classification:

- We need to optimize the decision based on
- cost involved using the confusion matrix
- Assuming cost $Cost_{ij} >= 0$ the cost of classifying Class i as class j and f_{ij} is the frequency class I is classified as class j.
- We need to build the decision system that minimizes the function: $\sum_{i=1}^{n}\sum_{j=1}^{n} Cost_{ij} f_{ij}$

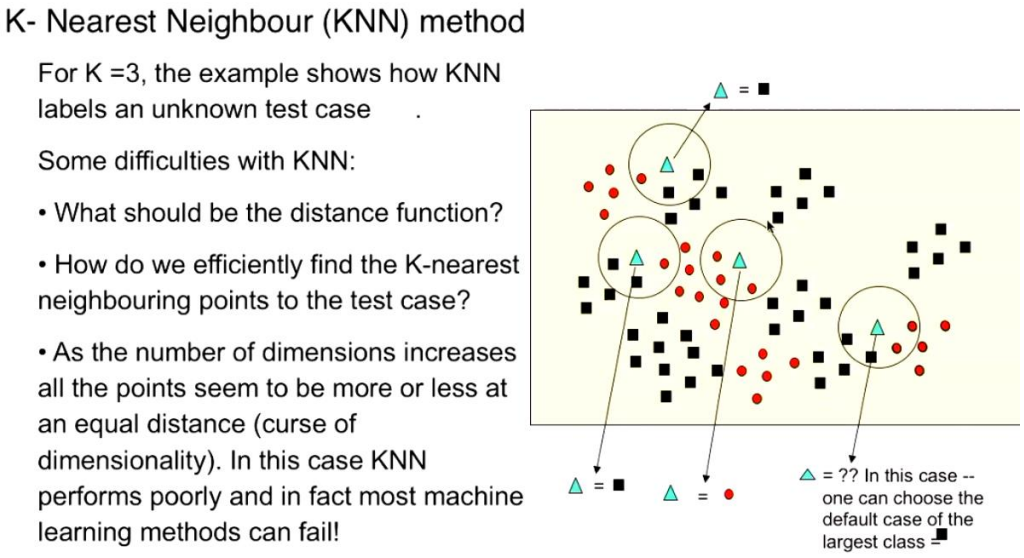
		Actual class			
		C1	C2	…	Cn
Predicted class	C1	Cost11	Cost21	…	Costn1
	C2	Cost12	Cost22	…	Costn2
	…	…	…	…	
	Cn	Cost1n	Cost2n	…	Costnn

Confusion Matrix		Actual class			
		C1	C2	…	Cn
Predicted class	C1	f11	ft21	…	fn1
	C2	f12	f22	…	ftn2
	…	…	…	…	
	Cn	f1n	f2n	…	fnn

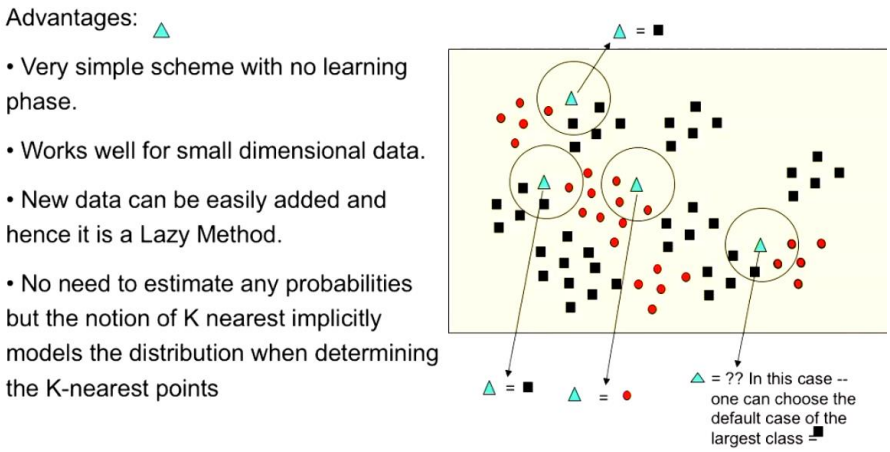
Lecture Set 7: KNN and Support Vector Machines

K- Nearest Neighbour (KNN)

- Geometric based method
- Simple technique and works well in many situations
- No learning involved
- Given a training data set with labelled information and test case, we find K training data points (records) nearest to the test case.
- The test case is labelled with the label of the most frequent label of the K nearest points to the test case.
- Given a test point, I will find the k nearest points to this test point in the training data. I know the labels of these K and so I pick the majority class.



We don't know green triangle label. Is it going to be a red or black label?



Some difficulties with KNN:

• What should be the distance function? Let p = <p1,p2,…,pn> and q = <q1,q2,…,qn> two points in an n-dimensional space.

$$\text{Euclidian distance}(p,q) = \|p-q\|^2 = \sqrt{\sum_{i=1}^{n}(pi-qi)^2}$$

$$\text{Minkowski distance}(p,q,k) = \sqrt[k]{\sum_{i=1}^{n}|pi-qi|^k}$$

$$\cos\_dist(p,q) = \frac{p.q}{\|p\| * \|q\|}$$

When

$k=1$ *is called* *L1 Norm*

$k=2$ *is called* *L2 Norm*

$$Radial\_Basis\_dist(\mathbf{p},\mathbf{q},\sigma) = \exp(-\frac{\|\mathbf{p}-\mathbf{q}\|^2}{2\sigma^2})$$

Minkowski is a k I choose, not the k distance k.

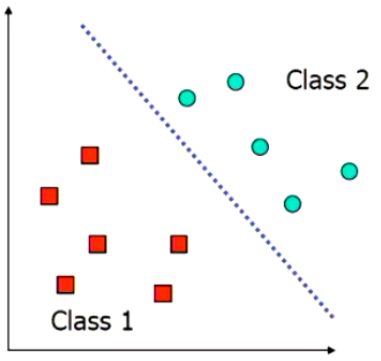
Cos_dist: angle between two vectors

Radial basis: find Euclid distance, then divide by constant and then negate and do expononetial.

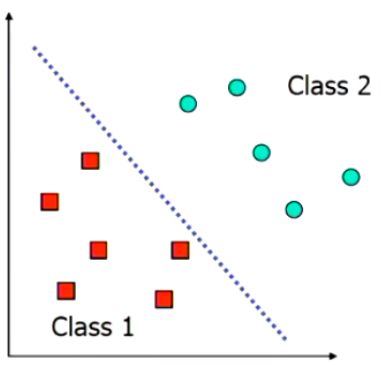
Support Vector Machines (SVM)

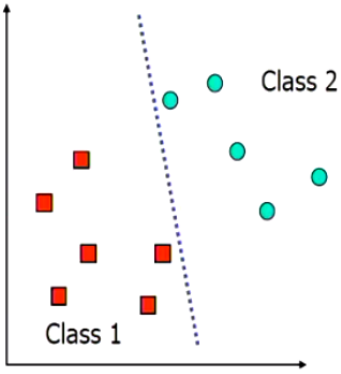
We can separate a linearly separable two-class using many decision boundaries. However, some decision boundaries are far better than the others. The goal of SVM is to find the optimal decision boundary.

Examples of a good Decision Boundary



Examples of Bad Decision Boundaries

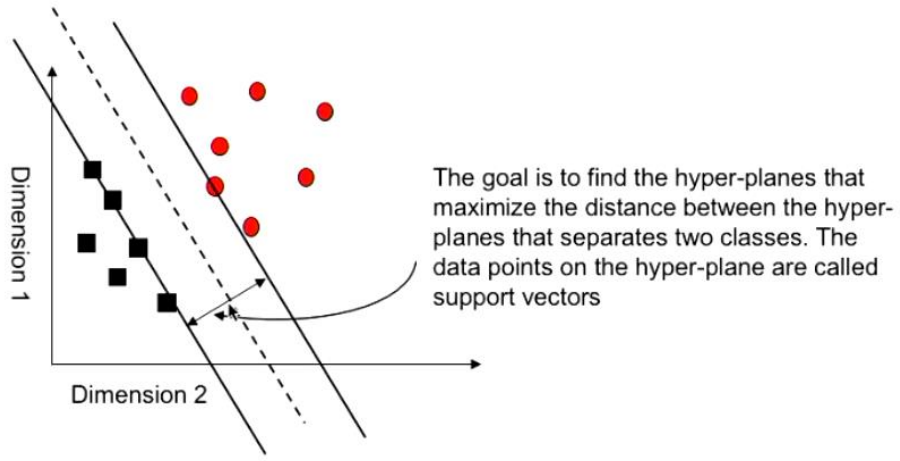




Find optimal hyper plane.

Support Vector Machines

Support vector machines is like Naïve Bayesian Classifier is a statistical machine learning Technique and it is also a geometric based method like KNN (K-nearest neighbour)



Want a hyper plane that maximally separate classes. These are called margin classifiers. Find hyper planes such that distance is maximal.